Math. Struct. in Comp. Science (2013), *vol.* 23, *pp.* 247–271. © Cambridge University Press 2013
doi:10.1017/S0960129512000102

Two-domain DNA strand displacement

LUCA CARDELLI

Microsoft Research, J. J. Thomson Avenue,
Cambridge, CB3 0FB, United Kingdom
Email: luca@microsoft.com

Received 6 October 2010; revised 23 June 2011

We investigate the computing power of a restricted class of DNA strand displacement structures: those that are made of double strands with nicks (interruptions) in the top strand. To preserve this structural invariant, we impose restrictions on the single strands they interact with: we consider only two-domain single strands consisting of one toehold domain and one recognition domain. We study fork and join signal processing gates based on these structures, and show that these systems are amenable to formalisation and mechanical verification.

1. Introduction

Among the many techniques being developed for molecular computing (Hagiya 2004), *DNA strand displacement* has been proposed as a mechanism for performing computation with DNA strands (Seelig *et al.* 2006; Fontana 2006). In most schemes, single-stranded DNA acts as *signals* and double-stranded (or more complex) DNA structures act as *gates*. Various circuits have been demonstrated experimentally (Yurke and Mills 2003; Seelig *et al.* 2006; Zhang *et al.* 2007; Zhang *et al.* 2011). The strand displacement mechanism is appealing because it is *autonomous* (Green *et al.* 2006): once signals and gates are mixed together, computation proceeds on its own without further intervention until the gates or signals are depleted (output is often read by fluorescence). The energy for computation is provided by the gate structures themselves, which are turned into inactive waste in the process. Moreover, the mechanism *only* requires DNA molecules: no organic sources, enzymes or transcription/translation ingredients are required, and the whole apparatus can be chemically synthesised and run in basic wet labs.

The main aims of this approach are to harness computational mechanisms that can operate at the molecular level and produce nano-scale structures under program control (Yin *et al.* 2008), and, somewhat separately, that can intrinsically interface to biological entities (Benenson *et al.* 2001). The computational structures that one may easily implement in this way (without some form of unbounded storage) vary from Boolean networks, to state machines, to Petri nets. The last two are particularly interesting because they take advantage of DNA's ability to encode symbolic information: they operate on DNA strands that represent abstract signals.

The fundamental mechanism in many of these schemes is *toehold mediated branch migration and strand displacement* (Yurke and Mills 2003; Zhang *et al.* 2007; Zhang *et al.* 2009), which implements a basic step of computation. It operates as shown in Figure 1,

https://doi.org/10.1017/S0960129512000102 Published online by Cambridge University Press

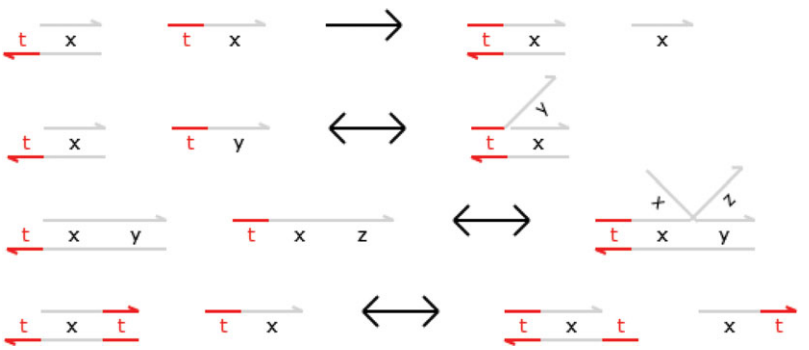


Fig. 1. (Colour online) Toehold-mediated DNA branch migration and strand displacement.

where each letter and corresponding segment represents a DNA *domain* (a sequence of nucleotides, C,G,T,A) and each DNA strand is seen as the concatenation of multiple domains. Single strands have an orientation; double strands are composed of two single strands with opposite orientation, where the bottom strand is the Watson–Crick, $C - G$, $T - A$, complement of the top strand. The 'short' domains hybridise (bind) *reversibly* to their complements, while the 'long' domains hybridise *irreversibly*; the exact critical length depends on the physical conditions. Distinct letters indicate domains that do not hybridise with each other.

In the first reaction of Figure 1, a short *toehold* domain t initiates binding between a double strand and a single strand. After the (reversible) binding of the toehold, the x domain of the single strand gradually replaces the top x strand of the double strand by *branch migration*. The branching point between the two top x domains performs a random walk that eventually leads to the *displacement* of the x strand. The final detachment of the top x strand makes the whole process essentially irreversible, because there is no toehold for the reverse reaction. The second reaction illustrates the case where the top domains do not match: then the toehold binds reversibly and no displacement occurs. The third reaction illustrates the more complicated situation where the top domains match only initially: the branch migration can proceed only up to a certain point and then must revert back to the toehold: hence no displacement occurs and the whole reaction reverts.

The fourth reaction illustrates a *toehold exchange*, where a branch migration (of strand tx) leads to a displacement (of strand xt), but where the whole process is reversible via a reverse toehold binding and branch migration. The first (irreversible) and fourth (reversible) reactions are the fundamental steps that can be composed to achieve computation by strand displacement.

2. Two-domain signals and gates

We now describe some DNA strand displacement structures that process abstract signals represented as DNA strands. Their function is to *join* input signals and *fork* output signals. To achieve compositionality so that gates can be composed arbitrarily into larger circuits, it is necessary to first fix the structure of the signals. Any given choice of signal structure

https://doi.org/10.1017/S0960129512000102 Published online by Cambridge University Press

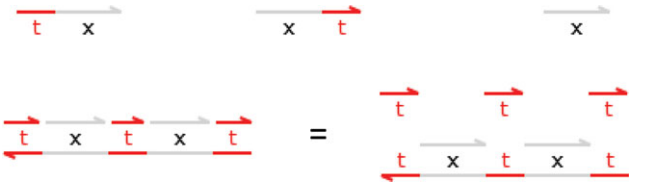


Fig. 2. (Colour online) Examples of allowable single and double strands: $tx, xt, x, \underline{t^\dagger x^\dagger t^\dagger x^\dagger t}$.

requires a different gate architecture, for example, for 4-domain signals (Soloveichik *et al.* 2010) and 3-domain signals (Cardelli 2011), that is, signals composed of 4 or 3 segments with different functions. Here we present a new, streamlined, architecture based on 2-domain signals, where the gates can be combined into arbitrary circuits (including loops), and where the waste products do not interfere with the active gates.

2.1. *Top-nicked double strands*

Double-stranded DNA (*dsDNA*) can have interruptions (*nicks*) on one strand, while remaining connected if the opposite strand has enough hold on the area around the nick. We have called such structures *nicked double-stranded DNA* (*ndsDNA*). This excludes any long overhangs or any protrusions from the double-strand. In particular, we work with *top-nicked double-strand*s, where all the nicks are on one strand (the top one by convention). A deviation from this simple structure happens fleetingly during branch migration, but all the initial and final species we use are ndsDNA.

We use t for short domains, x,y,z for long domains, and a,b,c for long domains that are meant to be privately used by some construction. We write, for example, tx for a single-stranded DNA (*ssDN* A) strand consisting of a toehold t followed by a domain x, and similarly for xt. We write, for example, $\underline{txy}$ for a fully complemented double strand consisting of a continuous strand txy at the top and its Watson–Crick complement at the bottom. Finally, we write $\underline{tx^\dagger y}$ to indicate the same double strand but with a nick at the top between x and y. In the figures, a nick is indicated by an arrowhead and a discontinuity. We assume that domains indicated by different letters are distinct, so that, for example, x does not hybridise with y, zy, yz, ty or yt.

Examples of allowable single and double strands are shown in Figure 2. To simplify our notation, we use an implicit equivalence illustrated in the bottom part of the figure. Suppose we start with a regular double strand, and we nick it at the top (bottom left). Long segments between nicks remain attached to the bottom strand, while short toehold segments can detach and reattach (bottom right). We regard these reversible states as equivalent; the notation $\underline{x^\dagger t^\dagger y}$ then indicates two equivalent situations, where the top t is either present or absent, and where t is implicitly exchanged with the environment. Hence, we can write $\underline{x^\dagger t^\dagger y}$ to indicate an open toehold between x and y, because the toehold is available (sometimes). In this way, we do not need to use separate notations for temporarily occluded and temporarily open toeholds, which we would have to regard as equivalent anyway (up to some kinetic occlusion effect).

https://doi.org/10.1017/S0960129512000102 Published online by Cambridge University Press

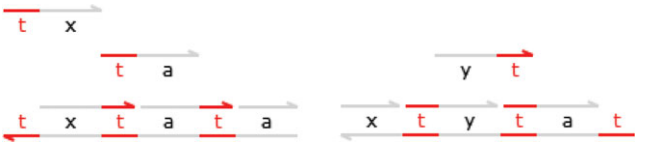


Fig. 3. (Colour online) Transducer T_{xy}: initial state (all strands except tx) plus input tx, leading to the reduction $T_{xy} \mid tx \rightarrow ty$.

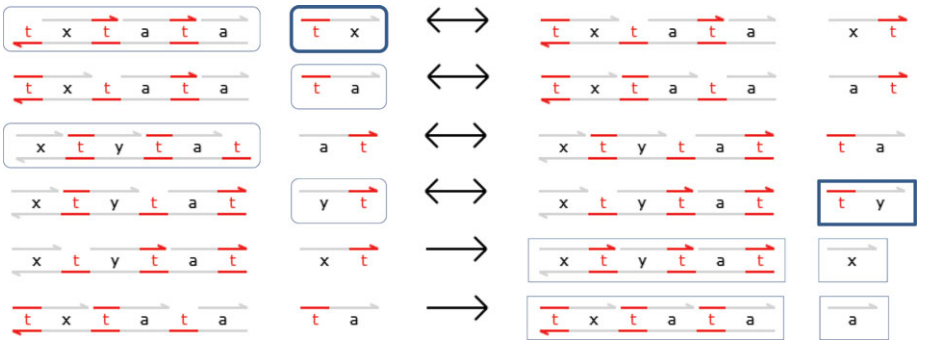


Fig. 4. (Colour online) Transducer $T_{xy} \mid tx \rightarrow ty$ reactions.

2.2. *Two-domain strand displacement gates*

Our gates are top-nicked dsDNA and our signals are two-domain ssDNA. This simple setup is more expressive than it might appear at first. For example, let us consider a single strand tx as encoding a *signal*, with the strand xt as its *cosignal*, and consider the problem of constructing a *signal transducer* T_{xy} (Figure 3) from a signal tx to a signal ty, with the *reduction* $T_{xy} \mid tx \rightarrow ty$, where $\mid$ is *parallel composition* of components, and final waste is discarded. All signals share the same toehold t, and are distinguished by the long domains x,y,z, and so on.

As shown in Figure 4, the input tx can initiate a signal/cosignal cascade of strand displacements in the left double-strand that after two toehold exchanges releases a *private* cosignal at (the segment a is privately used by the T_{xy} transducer, with a distinct a for each xy pair). The at cosignal then initiates a backward cascade in the right double strand that releases the desired output signal ty at the fourth reaction. The release of ty is reversible, but the gate is then locked down by the last two reactions. The locking down of the gate is also used to reabsorb the xt and ta strands by exploiting the $\underline{x}$ end of the right structure and the $\underline{a}$ end of the left structure. In the end, only *unreactive* (no exposed toeholds) dsDNA and ssDNA is left, except for the output ty.

Figure 4 lists the reactions of the transducer, with the initial structures from Figure 3 shown inside rounded rectangles and the final structures inside squared rectangles. Figure 5 shows the reactions of Figure 4 in the form of a reaction graph. The initial species are shown within a bold frame. Small squares indicate reactions: irreversible reactions have hollow arrowheads for their products and no arrowheads for their sources; reversible reactions have hollow arrowheads for their products and solid arrowheads for their sources.

https://doi.org/10.1017/S0960129512000102 Published online by Cambridge University Press

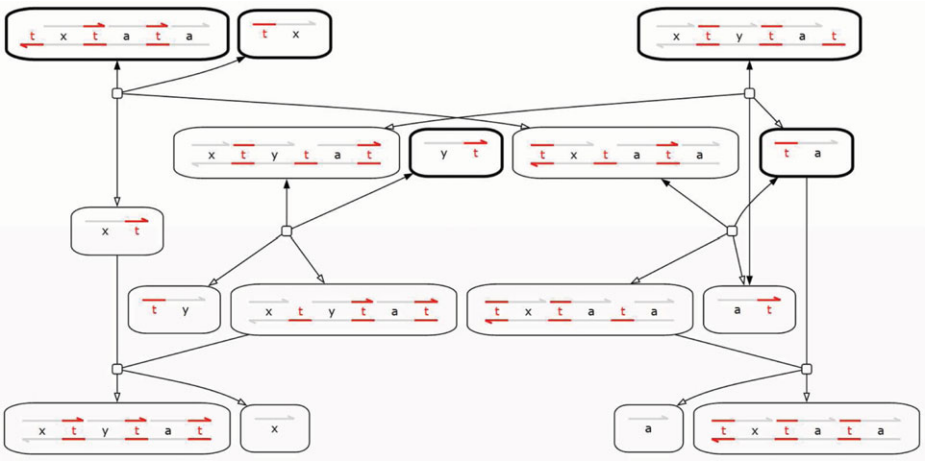


Fig. 5. (Colour online) Transducer $T_{xy} \mid tx \to ty$ reaction graph.

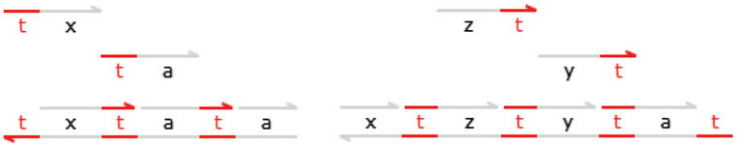


Fig. 6. (Colour online) Fork F_{xyz}: initial state (all strands except tx) plus input tx, leading to the reduction $F_{xyz} \mid tx \to ty \mid tz$.

The structures in Figure 3 can be written in the notation described above as

$$T_{xy} = \underline{t^\dagger x t^\dagger a t^\dagger a} \mid ta \mid \underline{x^\dagger t y^\dagger t a^\dagger t} \mid yt.$$

The auxiliary signal ta contains the private segment a, which uniquely joins the two halves of T_{xy} transducers, so we can assume that it will not interfere with other gates. However, the auxiliary cosignal yt contains a public segment y, which is necessary to release the output signal. It is therefore important to maintain an invariant that no other gate in the whole system absorbs yt, or, in general, any public cosignal, except in response to inputs, othewise those gates would be improperly triggered, and T_{xy} would be deprived of a necessary component. It is proper, and in fact necessary, to absorb public cosignals in response to inputs; for example, a T_{zy} transducer and a T_{xy} transducer may use'each other's' yt cosignal without problem.

The transducer T_{xy} can be extended easily to a fork gate F_{xyz} such that $F_{xyz} \mid tx \to ty \mid tz$, which releases two outputs from one input. This is shown in Figure 6, where the left half of the structure is the same as in T_{xy}. The construction of this gate (and similarly of T_{xy}, and all the following gates), remains valid when any of the inputs and ouputs are identified. That is, in this case, when $x = y$, $y = z$, $x = z$ or $x = y = z$. Some of these cases are in fact interesting: F_{xxx} amplifies the input signal until all such gates are exhausted. The fork gate can be extended to a catalytic gate C_{xyz} such that $C_{xyz} \mid tx \mid ty \to ty \mid tz$ (Figure 7). The right half of C_{xyz} is unchanged from F_{xyz}, except that yt is not required

https://doi.org/10.1017/S0960129512000102 Published online by Cambridge University Press

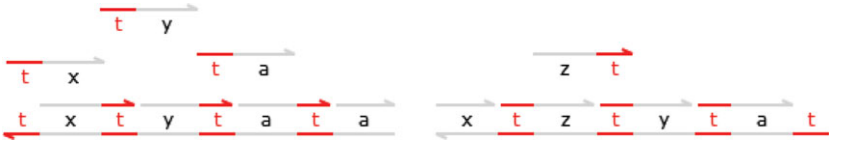


Fig. 7. (Colour online) Catalyst C_{xyz}: initial state (all strands except tx, ty) plus inputs tx, ty, leading to the reduction $C_{xyz} \mid tx \mid ty \rightarrow ty \mid tz$.

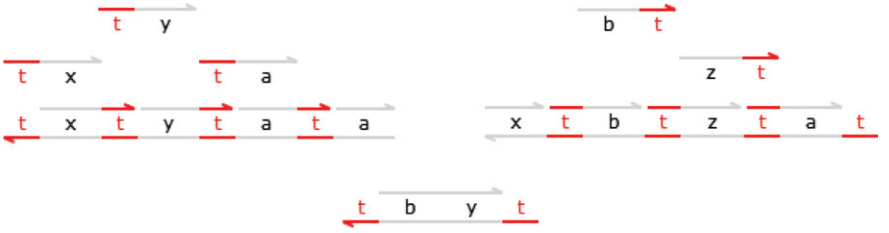


Fig. 8. (Colour online) Join J_{xyz}: initial state (all strands except tx, ty) plus inputs tx, ty, leading to the reduction $J_{xyz} \mid tx \mid ty \rightarrow tz$.

because it is produced by the left half. This gate, like the more general join gate discussed next, takes two inputs, but absorbs them only if both inputs are present (Soloveichik *et al.* 2010). If only the first input is present, it is returned to the soup by reversibility of strand displacement between tx and xt. A catalytic gate of the form C_{xyy} is called an autocatalyst.

We will now consider, in Figures 8 and 9, a binary join gate J_{xyz} such that

$$J_{xyz} \mid tx \mid ty \rightarrow tz$$

(the generalisation to additional outputs works as in the fork gate). Each distinct combination of xyz requires the choice of a distinct private domain connecting the two halves of the gate; this private domain can, however, be shared among a population of gates with the same input and output signals. The main new feature in this gate is the additional *cooperative displacement* structure $\underline{t^\dagger b y^\dagger t}$ (Zhang 2010; Zhang 2011) that absorbs a signal and a cosignal together, or neither separately. Without it, and without the bt, $\underline{tb}$ components, the join gate would leave behind a yt residual (all the other single strands, xt, zt, ta, are reclaimed). Hence, $\underline{t^\dagger b y^\dagger t}$ is a 'garbage collector' turning undesired active residuals to waste. It is triggered only after the release of a private strand tb,

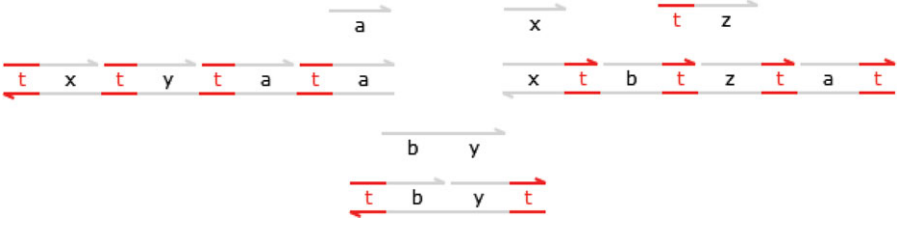


Fig. 9. (Colour online) Join reduction $J_{xyz} \mid tx \mid ty \rightarrow tz$: final state plus output tz.

https://doi.org/10.1017/S0960129512000102 Published online by Cambridge University Press

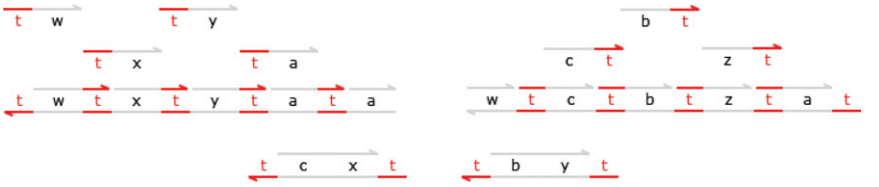


Fig. 10. (Colour online) 3-Join J_{wxyz}: initial state plus inputs tw, tx, ty, leading to output tz.

so the collector does not reclaim an extraneous cosignal yt before the join gate has committed to its inputs. Such an extraneous yt could come from a transducer T_{xy}, or from another join J_{uvy} (before any input) or J_{yuv} (after the first input) causing cross-gate interference, or even from within the same join, as in J_{xyy}. The join structure is easily generalised to any number of inputs; for example, Figure 10 shows a 3-input join with collectors.

Removing garbage is important because accumulated garbage slows down future reactions by imposing a growing reverse pressure on the desired direction of the reactions. We have designed all gates to remove all active garbage, but, before the introduction of the join gate, garbage removal did not require additional double strands. In practical cases, for a fixed computation, garbage collection can be ignored by initially taking some strands at high concentration, so that they kinetically overcome the effects of garbage accumulation (Soloveichik *et al.* 2010): the gate designs are then simpler. However, garbage accumulation also has an undesirable effect on the algebra of the next section since extra terms are then left after reductions, making it harder to compare circuits. In fact, the design of the algebra provided a strong motivation for investigating garbage-free realisations.

2.3. *Discussion*

2.3.1. *The double strand restrictions.* The restriction of allowing only ndsDNA structures has a number of potential advantages. The absence of any branching (*cf.* Soloveichik *et al.* (2010) and Cardelli (2011)) seems inherently more trouble-free than complex structures that can interact in unexpected ways through their protruding single-stranded parts. Here, all double-stranded structures are quiescent (apart from the receptive toeholds on the bottom strand) and only single-stranded components have hybridisation potential, eliminating the possibility that the gates themselves may polymerise, or self-interact. These structures also have a simple syntactical representation and simple reduction rules, which simplify formal verification. Nothing prevents us from devising precise syntax and reductions for more general structures (Phillips and Cardelli 2009), and there is no good reason in principle to avoid more complex structures if they work well. However, we have shown that our simplified structures already cover a surprising range of computation, and hence one can restrict the use of more complex structures to situations where they are actually needed, or where they perform better in some way.

2.3.2. *The single strand restrictions.* Our hybridised structures start as ndsDNA, but we have to ensure that they remain ndsDNA throughout the computation (apart from

https://doi.org/10.1017/S0960129512000102 Published online by Cambridge University Press

transients, that is, during branch migrations that either revert harmlessly or lead to strand displacements). This invariant puts constraints on the allowable single strands. First, single strands consisting only of long segments are inert because all the double strands are fully complemented (except for toeholds), so they can be ignored. A single strand of the form xty could bind to a double strand of the form $\underline{x^\dagger t^\dagger z}$, leading to a configuration that is stable and is not ndsDNA. Therefore, our single strands cannot contain substrands of the form xty, and we are left with single strands of the form, $x^n t^m$ or $t^n x^m$ or $t^n x^m t^p$. The third class could lead to stable configurations with two overlapping competing toeholds ($\underline{t^\dagger x^\dagger t^\dagger y^\dagger t}$ with txt and tyt) and hence are ruled out too. Multiple toeholds in sequence bind as stably as a long domain, so, for example, $xttt$ would be as bad as xty as it can lead to competing toeholds: $\underline{x^\dagger t^\dagger t^\dagger y}$ with xtt and tty. Hence, we do not allow consecutive toeholds in the top strands. Similarly, strands with consecutive long segments can lead to stable competition: txy and yzt over $\underline{t^\dagger xyz^\dagger t}$. As a result, we are just left with xt or tx, and the only remaining competition is between tx and xt over $\underline{t^\dagger x^\dagger t}$, where the stable structures are ndsDNA. A final case to consider is tx and yt over $\underline{t^\dagger xy^\dagger t}$: if a single strand is present, it only binds reversibly, and if both are present, they both bind stably and release xy, so the stable structures are always ndsDNA. In fact, $\underline{t^\dagger xy^\dagger t}$ is an important configuration that seems to add some power: without it, we can still implement garbage-collecting join gates, but apparently only by using more than one distinct toehold.

2.3.3. *The double strand restrictions, revisited.* Finally, we have to make sure that no reactive single strands other than t, tx, xt, plus the unreactive x and xy, are ever released from double strands during computation. This imposes another restriction on double strands: nicks should break the top strand into segments of two domains or less. Otherwise, the double strand $\underline{t^\dagger xty^\dagger t}$ could release a forbidden single strand xty in the presence of tx and yt. (We could still allow $\underline{t^\dagger xyz^\dagger t}$, but it would be unreactive.) Hence, we are left with allowable double strands that are nicked concatenations of the double-stranded elements $\underline{t}$, $\underline{x}$, $\underline{tx}$, $\underline{xt}$, $\underline{xy}$.

2.4. *Multiple toeholds*

In order to allow for reversible binding, toeholds cannot exceed a certain critical length that depends on physical conditions. Reversible toehold binding is in turn necessary to revert mismatches, as described in Section 1. Hence, in each physical condition we can have only a fixed finite number of distinct toeholds (Soloveichik *et al.* 2010) (in contrast, we can have, in principle, an unbounded number of distinct long domains). It is possible to use the available distinct toeholds to some advantage, for example, to separate the possible successful matches in different categories, hence reducing the kinetic effects of unsuccessful matches: these have no logical consequence but can slow down operation. Another use for distinct toeholds is in garbage collection to make sure that garbage collectors do not interfere with normal operation.

We now discuss an alternative design for join gates that uses two distinct toeholds, with the additional feature that trimolecular reactions (cooperative displacements, as

https://doi.org/10.1017/S0960129512000102 Published online by Cambridge University Press

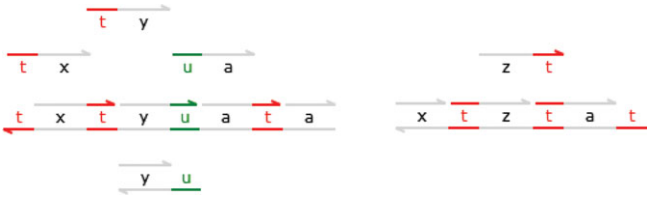


Fig. 11. (Colour online) Join $J'_{xyz} \mid tx \mid ty \rightarrow tz$: initial state with inputs tx, ty.

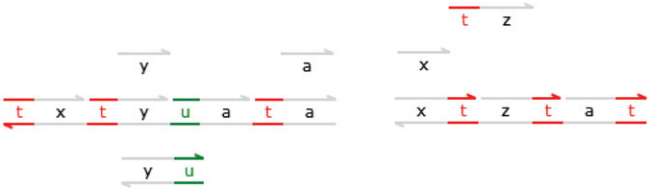


Fig. 12. (Colour online) Join $J'_{xyz} \mid tx \mid ty \rightarrow tz$: final state with output tz.

in Figure 8) are no longer required. However, we shall see that this design does not generalise as easily to higher numbers of inputs. The left half of this alternative join gate, J'_{xyz} (Figures 11 and 12), is the same as the left half of Figure 8, except that a distinct toehold u is used for the auxiliary strand ua. As a consequence, the yu cosignal can be garbage collected immediately after release because there is no interference with yt cosignals elsewhere. Moreover, the right half of the gate can be simplified because it no longer needs to trigger the collection of yu. The collector itself, $\underline{y^{\dagger}u}$, is simpler and does not rely on trimolecular reactions.

Note that all J' join gates can share the same u toehold, because the appearance of a zu cosignal indicates that some gate with second input tz (say, either J'_{xz-} or J'_{yz-}) has received both inputs. In such a situation it is fine to collect one zu strand permanently because one gate has received its inputs; it does not matter which collector removes zu – the two collectors are in fact identical. Therefore, for this gate design, we need a total of two distinct toeholds: t and u.

Unfortunately, the collectors for gates with three or more inputs become more complex, and, in fact, they closely resemble the collectors for the 3-domain gates used in Cardelli (2011). For a 3-way join J'_{wxyz} with inputs w, x, y and output z, the non-collecting part is again similar to J_{wxyz} (Figure 10), apart from ua and corresponding domains. Here we need to design a collector for the cosignals xt and yu that are displaced by the second and third inputs, where collection should be triggered by the emergence of yu (wt arising from the first input is reclaimed as before). Unfortunately, the simple-minded solution for such a collector shown in Figure 13 releases a ty signal, which is an input signal. Introducing a collector for this ty results in the collection of ty signals even when the join gate does not have all its inputs.

https://doi.org/10.1017/S0960129512000102 Published online by Cambridge University Press

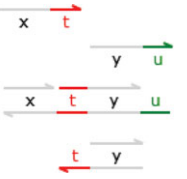


Fig. 13. (Colour online) Bad collector for 3-input join J'_{wxyz}.

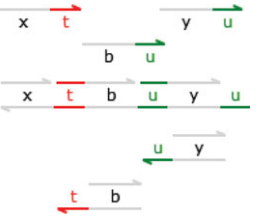


Fig. 14. (Colour online) Good collector for 3-input join J'_{wxyz}.

A solution (Figure 14) is to introduce an extra private domain b to break ty into tb and uy, which can both be collected without inteference. This pattern extends to collectors for n-join gates, with t and u toeholds alternating on the main collector double strand, so that no ordinary signals are released and no ordinary cosignals are absorbed (for example, $w^\dagger tc^\dagger ux^\dagger tb^\dagger uy^\dagger u$ for a 4-join J'_{vwxyz} collector main double strand, with private b,c).

These multiple-toehold collectors are therefore significanly more complex than the ones in the previous sections. We will only consider single-toehold gate designs in the rest of this paper; another example of a single-toehold design can be found in Qian and Winfree (2011).

2.5. *Irreversible output release*

In our two-domain gates, the final irreversible steps that commit the execution of a gate happen only after the outputs are reversibly released. That means that the released outputs from a gate G can revert before the irreversible commit for G, and cause the inputs of G to be released again, as if G 'had never fired'. However, the outputs of G, even if only temporarily released, can have a catalytic effect downstream (they can be used as inputs by other gates that re-emit them from their outputs). In such a catalytic network, there can be a sequence of events where, in the presence of its inputs, the gate G 'does not fire' because neither it nor its inputs are consumed, but it still causes the permanent activation of other gates downstream. For example,

$$tx \mid tu \mid T_{xy} \mid T_{xz} \mid C_{uyw}$$

can produce either

$$T_{xz} \mid ty \mid tw$$

or

$$tu \mid T_{xy} \mid C_{uyw} \mid tz$$

https://doi.org/10.1017/S0960129512000102 Published online by Cambridge University Press

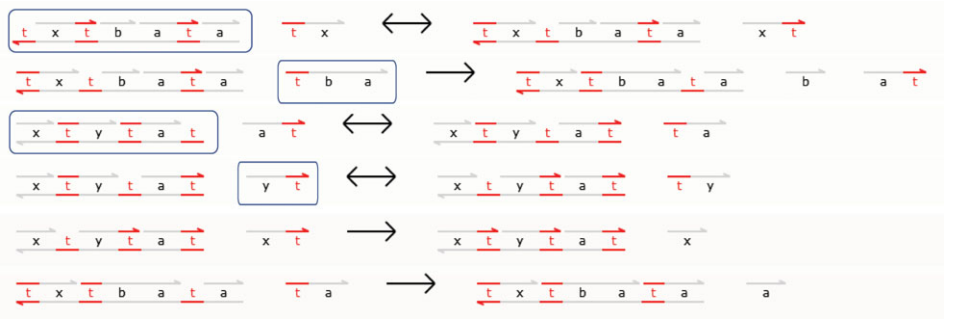


Fig. 15. (Colour online) Transducer T'_{xy} with irreversible output release.

depending on which T_{x-} gate is irreversibly activated by tx, but it can also produce

$$T_{xy} \mid ty \mid tw \mid tz$$

if T_{xy} is first reversibly activated. The probability of such a sequence of events can be arbitrarily reduced by introducing additional transducers ahead of the catalytic gate,

$$T_{xy_1} \mid T_{y_1 y_2} \mid \ldots \mid T_{y_n y} \mid C_{uyw},$$

so as to increase the probability that an irreversible step will happen during that sequence. This shows that two-domain gates can emulate irreversible output release up to arbitrary precision (but not exactly). Another way to approximate irreversible output release within a single gate is, for example, to use increasing amounts of the *ta* strand in Figure 3, to make the sixth reaction of Figure 4 increasingly likely.

There are gate designs where an irreversible step happens *necessarily* after all the inputs are absorbed and before (or at the same time that) all the outputs are released. This is in fact the case for the designs in Soloveichik *et al.* (2010) for four-domain gates and in Cardelli (2011, Figure 9) for three-domain gates. But this does not seem possible with our basic two-domain operations because each operation is either irreversible and does not release a (co)signal, or it releases a (co)signal but is reversible (see Figure 16 in the next section), so an irreversible step cannot cause a later release of a (co)signal.

We will now discuss a design for a modified ndsDNA transducer that irreversibly releases its output. More precisely, it is such that if the output is ever released, the input cannot be released back, and hence the gate commits on its input (the output is still initially released reversibly, but if it is released, it eventually has to be released irreversibly). The modifications extend easily to gates with multiple inputs and outputs because they affect only the private signals between the left and right halves. Figure 15 shows the reactions of the modified transducer

$$T'_{xy} = \underline{t^\dagger x t^\dagger b^\dagger a t^\dagger a} \mid tba \mid \underline{x^\dagger t y^\dagger t a^\dagger t} \mid yt,$$

where the initial strands forming the transducer are highlighted. When compared with the transducer reactions in Figure 4, the second reaction has turned from reversible to irreversible. This is achieved by an additional private domain $\underline{b}$ in the left double strand, and by a corresponding three-domain strand that binds irreversibly because of a nick after

https://doi.org/10.1017/S0960129512000102 Published online by Cambridge University Press

$\underline{b}$ and releases the same *at* cosignal as before. The right half of the transducer receiving that cosignal is unchanged. That second, irreversible, reaction happens before the output is released, and even if the output binds back temporarily before the final gate lockdown (provided as before by the fifth and sixth reactions), the output cannot cause the input to be released again.

This design uses three-domain strands, but since they include private domains that limit their interactions, the ndsDNA structure happens to be preserved during execution. Moreover, the top strands of double stranded structures still all consist of one or two contiguous domains, hence no new three domain strands can be released. The strands used for public signals are still two-domain strands; hence these modified gates are signal-compatible with the other two-domain gates, and can be used tactically where irreversible output release is logically necessary. However, for simplicity, we do not consider three-domain strands in the algebraic framework of the next section.

3. Nick algebra

In this section we provided a formal framework in which we can perform calculations about the evolution of systems of top-nicked double strands. *Domains* are taken either from a finite set of *short domains* (*toeholds*) or from an unbounded set of *long domains* ranged over by x,y,z and a,b,c. Designs based on a single toehold can be easily adapted to multiple toeholds to increase binding discrimination and efficiency, but the converse is problematic: designs based on distinct toeholds may fail if the toeholds are then identified. Here we require only a single distinguished toehold, always indicated by the constant t, but it would be easy to generalise the discussion to multiple toeholds.

An infix operator '.' may be used to concatenate domains into single strands; this is often omitted, particularly because all our single-strands have the form $t.x$ or $x.t$, which are then usually written tx and xt (unless we wish to use long identifiers for domains). Single strands t, x, and $x.y$ remain implicit 'waste': they are implicitly removed in reductions, and therefore do not appear in the syntax.

Double strands are written underlined. We use an infix operator '$\underline{\dagger}$' to represent a 'nick' on the top strand of a double-stranded sequence, an infix operator '$\underline{.}$' (often omitted) to represent the unbroken concatenation of top and bottom strands, and ϕ for the empty double strand. The segments between nicks are only single or pair combinations of toeholds and domains.

A soup U is a finite multiset of single and double strands, with multiset union indicated by ' | ', and with a notation $(\nu x)U$ for domain isolation. The latter indicates that x is not used outside of U: this allows us to declare private domains locally, and to combine constructions compositionally. In practice, it means simply that all the domains indicated by ν must be chosen distinct when a global system is fixed for execution: the algebraic laws for $(\nu x)U$ encode such a guarantee. We also use U^n as an abbreviation for n copies of U in parallel (|). The resulting algebra is our nick algebra, which is strictly a subset of the DSD (DNA Strand Displacement) language (Phillips and Cardelli 2009). The grammar of terms is given in Definition 3.1, with terminal symbols t, ϕ and non-terminal symbols x

https://doi.org/10.1017/S0960129512000102 Published online by Cambridge University Press

(ranging over domains), S, $\underline{D}$, U, and with '$\mid$' for syntactic alternatives; parentheses are also liberally used for precedence.

Definition 3.1 (Term syntax).

$S ::= t.x \mid x.t$	Single strand
$\underline{D} ::= \phi \mid \underline{t} \mid \underline{x} \mid \underline{t.x} \mid \underline{x.t} \mid \underline{x.x} \mid \underline{D}^\dagger \underline{D}$	Double strand
$U ::= S \mid \underline{D} \mid U \mid U \mid (\nu x)U$	Soup

The set of *public domains* $pd(U)$ is the inductively defined set of those domains not bound by ν in U. In particular:

$$pd(t.x) = pd(x.t) = pd(\underline{x}) = pd(\underline{t.x}) = pd(\underline{x.t}) = \{x\}$$

$$pd(\underline{x.y}) = \{x, y\}$$

$$pd(\underline{t}) = pd(\phi) = \{\}$$

$$pd((\nu x)U) = pd(U) - \{x\}.$$

Then, $U\{y/x\}$ is the substitution of y for x in U, with the representative cases

$$\begin{aligned}
t\{y/x\} &= t \\
x\{y/x\} &= y \\
z\{y/x\} &= z && \text{for } z \neq x \\
((\nu z)U)\{y/x\} &= (\nu z)(U\{y/x\}) && \text{for } z \notin \{x, y\} \\
((\nu x)U)\{y/x\} &= (\nu x)U \\
((\nu y)U)\{y/x\} &= (\nu z)(U\{z/y\}\{y/x\}) && \text{for a } z \notin pd(U) \cup \{x, y\}.
\end{aligned}$$

Algebraic equality (a binary congruence relation over the term syntax) is indicated simply by $=$ and is axiomatised below with the monoid laws of $(\phi, ^\dagger)$, the commutative monoid laws of $(\phi, \mid)$, and the scoping laws of $(\nu x)U$ (Milner 1999).

Definition 3.2 (Algebraic equality). $=$ is an equivalence relation, and:

$$\begin{aligned}
\underline{D_1} = \underline{D_2},\ \underline{D_3} = \underline{D_4} &\Rightarrow \underline{D_1}^\dagger \underline{D_3} = \underline{D_2}^\dagger \underline{D_4} \\
U_1 = U_2,\ U_3 = U_4 &\Rightarrow U_1 \mid U_3 = U_2 \mid U_4 \\
U_1 = U_2 &\Rightarrow (\nu x)U_1 = (\nu x)U_2
\end{aligned}$$

$$\begin{aligned}
\underline{D_1}^\dagger(\underline{D_2}^\dagger \underline{D_3}) &= \underline{(D_1}^\dagger \underline{D_2)}^\dagger \underline{D_3} \\
\phi^\dagger \underline{D} &= \underline{D}^\dagger \phi = \underline{D}
\end{aligned}$$

$$\begin{aligned}
U_1 \mid (U_2 \mid U_3) &= (U_1 \mid U_2) \mid U_3 \\
U_1 \mid U_2 &= U_2 \mid U_1 \\
\phi \mid U &= U \mid \phi = U
\end{aligned}$$

https://doi.org/10.1017/S0960129512000102 Published online by Cambridge University Press

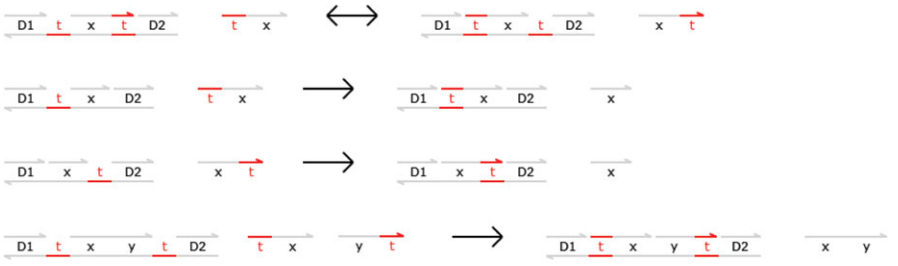


Fig. 16. (Colour online) The basic reactions (D1,D2 are arbitrary or empty double strands).

$$
\begin{aligned}
(\nu x)U &= (\nu y)(U\{y/x\}) && \text{if } y \notin pd(U)\\
(\nu x)\phi &= \phi\\
(\nu x)(U_1 \mid U_2) &= U_1 \mid (\nu x)U_2 && \text{if } x \notin pd(U_1)\\
(\nu x)(\nu y)U &= (\nu y)(\nu x)U.
\end{aligned}
$$

Note that $(\nu x)(\nu x)U = (\nu x)U$ is derivable. As an example of the use of the isolation operation, consider that it is always possible to bring all the ν prefixes to the top level by making all the private domains distinct:

$$(\nu x)tx \mid (\nu x)tx = (\nu x)tx \mid (\nu y)ty = (\nu x)(\nu y)(tx \mid ty).$$

This means that conflicts between local definitions can be resolved globally, while allowing local definition to be combined without the need to consider global conflicts.

The *reduction* relation $U_1 \to U_2$ describes a single step of system evolution; it is the smallest binary relation on U satisfying the rules below, where $\leftrightarrow$ stands for two reduction rules in opposite directions. Its symmetric and transitive closure $U_1 \to^* U_2$ describes multi-step system evolution. In the reduction rules, the *single-stranded waste* (t, x, xy) is automatically removed because it can be immediately identified as waste (as a consequence, the single strands t, x, xy need not be included in the syntax). Alternatively, we could have made the single-stranded waste explicit and introduced separate rules to remove it. The double-stranded waste instead has a special degradation rule because it requires a check over the whole double strand. The four basic reactions (exchange, left and right coverage, and cooperation) are shown in Figure 16.

Definition 3.3 (Reduction).

$$
\begin{aligned}
\underline{D_1{}^\dagger t^\dagger x t^\dagger D_2} \mid tx &\leftrightarrow \underline{D_1{}^\dagger tx^\dagger t^\dagger D_2} \mid xt && \text{Exchange}\\
\underline{D_1{}^\dagger t^\dagger x^\dagger D_2} \mid tx &\to \underline{D_1{}^\dagger tx^\dagger D_2} && \text{Left coverage}\\
\underline{D_1{}^\dagger x^\dagger t^\dagger D_2} \mid xt &\to \underline{D_1{}^\dagger xt^\dagger D_2} && \text{Right coverage}\\
\underline{D_1{}^\dagger t^\dagger xy^\dagger t^\dagger D_2} \mid tx \mid yt &\to \underline{D_1{}^\dagger tx^\dagger yt^\dagger D_2} && \text{Cooperation}
\end{aligned}
$$

https://doi.org/10.1017/S0960129512000102 Published online by Cambridge University Press

$$\begin{array}{rcll} \underline{D} \to \phi & \text{if } \underline{D} \text{ not } \textit{reactive} & & \text{Waste} \\ U_1 \to U_2 & \Rightarrow & U_1 \mid U \to U_2 \mid U & \text{Dilution} \\ U_1 \to U_2 & \Rightarrow & (\nu x)U_1 \to (\nu x)U_2 & \text{Isolation} \\ U_1 = U_2, U_2 \to U_3, U_3 = U_4 & \Rightarrow & U_1 \to U_4. & \text{Well-mixing} \end{array}$$

A double strand $\underline{D}$ is *reactive* if it can react in some context; that is, by the first four rules. Hence it must be of the form

$$\underline{D_1{}^\dagger t^\dagger x t^\dagger D_2}$$
$$\underline{D_1{}^\dagger t x^\dagger t^\dagger D_2}$$
$$\underline{D_1{}^\dagger t^\dagger x^\dagger D_2}$$
$$\underline{D_1{}^\dagger x^\dagger t^\dagger D_2}$$

or

$$\underline{D_1{}^\dagger t^\dagger x y^\dagger t^\dagger D_2}.$$

Examples of the unreactive (waste) double strands are thus $\underline{t}$, $\underline{x}$, $\underline{xt}$, $\underline{tx}$, $\underline{xy}$, $\underline{t^\dagger t}$, $\underline{t^\dagger tx}$, $\underline{xt^\dagger t}$, $\underline{xt^\dagger ty}$ and $\underline{xt^\dagger t^\dagger ty}$. The Waste rule is really a convenience to simplify results of calculations; more generally, as commonly done in process algebra, one would instead eliminate unreactive components through an observational equivalence (Milner 1999). Two standard rules of process algebra reflect the assumption of diluted and well-mixed chemical soups (Berry and Boudol 1989; Cardelli 2011). The Dilution rule says that adding some U to a soup U_1 does not make it impossible for U_1 to reduce as before (although U may enable additional reductions). The Well-mixing rule connects the reduction relation with the associativity/commutativity of $=$, implying that it is possible to mix the soup in order to bring 'distant' terms in syntactic contact (the reduction rules syntactically require interacting terms to be next to each other).

4. Correctness

If $U_1 \to^* U_2$, then U_1 may reduce to U_2, but it may also reduce to something else since $\to^*$ is a relation. When $U_1 \to^* U_2$ is used to state a correctness property of system reduction, we say that this is a *may-correctness* property: the system starting from U_1 *may* reduce to U_2, but it may also wander into a different section of state space and never be able to get to U_2 from there. A stronger property is *will-correctness*, indicated by $U_1 \to^\forall U_2$, and defined as

$$\forall U,\ U_1 \to^* U \ \Rightarrow\ U \to^* U_2.$$

This means that although U_1 may wander into some U in some part of the state space, it *will* always find a path to U_2 from there (it *cannot avoid* finding a path to U_2). If $U_1 \to^\forall U_2$ and U_2 is the only terminal state, then we can say that U_1 *must reduce* to U_2. But will-correctness does not imply that reduction necessarily terminates, and, in

https://doi.org/10.1017/S0960129512000102 Published online by Cambridge University Press

particular, if $U \rightarrow^{\forall} U$, we can say that U is *reversible*. Since $U_1 \rightarrow^* U_1$ holds by reflexivity, will-correctness implies may-correctness. All these properties are really examples of a large class of reachability properties that could be expressed in a temporal logic.

It is convenient in the next examples and proofs to use a more pictographic notation for nick algebra expressions to highlight the positions of the toeholds. We use the abbreviations in the following definition ($\underline{\ }^\dagger$ is still needed for $\underline{x^\dagger y}$).

Definition 4.1 (Two-domain pictograms).

$\ulcorner x$	for tx	*Signal*
$x\urcorner$	for xt	*Cosignal*
$\underline{D\ulcorner x}$	for $\underline{D^\dagger tx}$ (including $D = \phi$)	Bound signal
$\underline{x\urcorner D}$	for $\underline{xt^\dagger D}$ (including $D = \phi$)	Bound cosignal
$\underline{D \smile D'}$	for $\underline{D^\dagger t^\dagger D'}$ (including $D = \phi$ or $D' = \phi$)	Bottom toehold

For example, the transducer from Figure 3 can be written as:

$$\underline{t^\dagger x t^\dagger a t^\dagger a} \mid ta \mid \underline{x^\dagger t y^\dagger t a^\dagger t} \mid yt \qquad \text{(explicit notation)}$$

or

$$\underline{\smile x\urcorner a\urcorner a} \mid \ulcorner a \mid \underline{x\ulcorner y\ulcorner a\smile} \mid y\urcorner. \qquad \text{(pictogram notation)}$$

4.1. *May-correctness for populations*

We now show that the transducer *may* work correctly. Because of their chemical origin, all components come in populations of identical molecules, and any private domain can only be private to a population, and not to an individual molecule. Hence, we need to show that a populations of transducers, all sharing the same private domain, *may* map an input population to a desired output population. We use U^n as an abbreviation for n copies of U in parallel (|).

Proposition 4.2 (Transducer T^n_{xy} may-correctness). Let

$$T^n_{xy} = (\nu a)((\underline{\smile x\urcorner a\urcorner a} \mid \ulcorner a \mid \underline{x\ulcorner y\ulcorner a\smile} \mid y\urcorner)^n).$$

Then

$$T^n_{xy} \mid \ulcorner x^n \rightarrow^* \ulcorner y^n.$$

Proof. Let

$$T_{xay} = \underline{\smile x\urcorner a\urcorner a} \mid \ulcorner a \mid \underline{x\ulcorner y\ulcorner a\smile} \mid y\urcorner$$

for $a \neq x, y$ so that

$$T^n_{xy} = (\nu a)((T_{xay})^n).$$

We first show that

$$T_{xay} \mid \ulcorner x \rightarrow^* \ulcorner y.$$

https://doi.org/10.1017/S0960129512000102 Published online by Cambridge University Press

We have:

$$\begin{aligned}
T_{xay} \mid \ulcorner x &= \underline{\smile x \urcorner a \urcorner a} \mid \ulcorner a \mid \underline{x \ulcorner y \ulcorner a \smile} \mid y \urcorner \mid \ulcorner x \\
&\leftrightarrow \underline{\ulcorner x \smile a \urcorner a} \mid \ulcorner a \mid \underline{x \ulcorner y \ulcorner a \smile} \mid y \urcorner \mid x \urcorner \\
&\leftrightarrow \underline{\ulcorner x \ulcorner a \smile a} \mid \underline{x \ulcorner y \ulcorner a \smile} \mid y \urcorner \mid x \urcorner \mid a \urcorner \\
&\leftrightarrow \underline{\ulcorner x \ulcorner a \smile a} \mid \underline{x \ulcorner y \smile a \urcorner} \mid y \urcorner \mid x \urcorner \mid \ulcorner a \\
&\rightarrow \underline{\ulcorner x \ulcorner a \ulcorner a} \mid \underline{x \ulcorner y \smile a \urcorner} \mid y \urcorner \mid x \urcorner \\
&\rightarrow \underline{x \ulcorner y \smile a \urcorner} \mid y \urcorner \mid x \urcorner \\
&\leftrightarrow \underline{x \smile y \urcorner a \urcorner} \mid x \urcorner \mid \ulcorner y \\
&\rightarrow \underline{x \urcorner y \urcorner a \urcorner} \mid \ulcorner y \\
&\rightarrow \ulcorner y.
\end{aligned}$$

Hence,

$$(T_{xay} \mid \ulcorner x)^n \rightarrow_r^* y^n$$

by induction,

$$(T_{xay})^n \mid \ulcorner x^n \rightarrow_r^* y^n$$

by associativity,

$$(\nu a)((T_{xay})^n \mid \ulcorner x^n) \rightarrow^* (\nu a) \ulcorner y^n$$

by isolation, and

$$T_{xy}^n \mid \ulcorner x^n \rightarrow_r^* y^n$$

by ν-equivalence and the definition of T_{xy}^n. □

We can similarly check the may-correctness of fork and join gates as follows.

Proposition 4.3 (Fork F_{xyz}^n may-correctness). Let

$$F_{xyz}^n = (\nu a)((\underline{\smile x \urcorner a \urcorner a} \mid \ulcorner a \mid \underline{x \ulcorner z \ulcorner y \ulcorner a \smile} \mid z \urcorner \mid y \urcorner)^n).$$

Then

$$F_{xyz}^n \mid \ulcorner x^n \rightarrow^* \ulcorner y^n \mid \ulcorner z^n.$$

Proof. Let

$$F_{xayz} = \underline{\smile x \urcorner a \urcorner a} \mid \ulcorner a \mid \underline{x \ulcorner z \ulcorner y \ulcorner a \smile} \mid z \urcorner \mid y \urcorner$$

for $a \neq x, y, z$ so that

$$F_{xyz}^n = (\nu a)((F_{xayz})^n).$$

We first show that

$$F_{xayz} \mid \ulcorner x \rightarrow^* \ulcorner y \mid \ulcorner z.$$

https://doi.org/10.1017/S0960129512000102 Published online by Cambridge University Press

We have:

$$\begin{aligned}
F_{xayz} \mid \ulcorner x = \underline{\smile x \urcorner a \urcorner a} \mid \ulcorner a \mid \underline{x \ulcorner z \ulcorner y \ulcorner a \smile} \mid z \urcorner \mid y \urcorner \mid x \urcorner \\
\leftrightarrow \underline{\ulcorner x \smile a \urcorner a} \mid \ulcorner a \mid \underline{x \ulcorner z \ulcorner y \ulcorner a \smile} \mid z \urcorner \mid y \urcorner \mid x \urcorner \\
\leftrightarrow \underline{\ulcorner x \ulcorner a \smile a} \mid \underline{x \ulcorner z \ulcorner y \ulcorner a \smile} \mid z \urcorner \mid y \urcorner \mid x \urcorner \mid a \urcorner \\
\leftrightarrow \underline{\ulcorner x \ulcorner a \smile a} \mid \underline{x \ulcorner z \ulcorner y \smile a \urcorner} \mid z \urcorner \mid y \urcorner \mid x \urcorner \mid \ulcorner a \\
\rightarrow \underline{\ulcorner x \ulcorner a \ulcorner a} \mid \underline{x \ulcorner z \ulcorner y \smile a \urcorner} \mid z \urcorner \mid y \urcorner \mid x \urcorner \\
\rightarrow \underline{x \ulcorner z \ulcorner y \smile a \urcorner} \mid z \urcorner \mid y \urcorner \mid x \urcorner \\
\leftrightarrow \underline{x \ulcorner z \smile y \urcorner a \urcorner} \mid z \urcorner \mid x \urcorner \mid \ulcorner y \\
\leftrightarrow \underline{x \smile z \urcorner y \urcorner a \urcorner} \mid x \urcorner \mid \ulcorner y \mid \ulcorner z \\
\rightarrow \underline{x \urcorner z \urcorner y \urcorner a \urcorner} \mid \ulcorner y \mid \ulcorner z \\
\rightarrow \mid \ulcorner y \mid \ulcorner z.
\end{aligned}$$

Hence,

$$(F_{xayz} \mid \ulcorner x)^n \rightarrow^* (\ulcorner y \mid \ulcorner z)^n$$

by induction,

$$(F_{xayz})^n \mid \ulcorner x^n \rightarrow_{\ulcorner} y^n \mid \ulcorner z^n$$

by associativity,

$$(\nu a)((F_{xayz})^n \mid \ulcorner x_n) \rightarrow^* (\nu a)(\ulcorner y^n \mid \ulcorner z^n)$$

by isolation, and

$$F^n_{xyz} \mid \ulcorner x^n \rightarrow^* \ulcorner y^n \mid \ulcorner z^n$$

by ν-equivalence and the definition of F^n_{xyz}. □

Proposition 4.4 (Join J^n_{xyz} may-correctness). Let

$$J^n_{xyz} = (\nu a)(\nu b)((\underline{\smile x \urcorner y \urcorner a \urcorner a} \mid \ulcorner a \mid \underline{x \ulcorner b \ulcorner z \ulcorner a \smile} \mid b \urcorner \mid z \urcorner \mid \underline{\smile b^\dagger y \smile})^n).$$

Then

$$J^n_{xyz} \mid \ulcorner x^n \mid \ulcorner y^n \rightarrow^* \ulcorner z^n.$$

Proof. Let

$$J_{xyaz} = \underline{\smile x \urcorner y \urcorner a \urcorner a} \mid \ulcorner a \mid \underline{x \ulcorner b \ulcorner z \ulcorner a \smile} \mid b \urcorner \mid z \urcorner \mid \underline{\smile b^\dagger y \smile}$$

for $a \neq x, y, z$ so that

$$J^n_{xyz} = (\nu a)((J_{xyaz})^n).$$

We first show that

$$J_{xyaz} \mid \ulcorner x \mid \ulcorner y \rightarrow^* \ulcorner z.$$

https://doi.org/10.1017/S0960129512000102 Published online by Cambridge University Press

We have:

$$\begin{aligned}
J_{xyaz} \mid \ulcorner x \mid \ulcorner y &= \underline{\smile x \urcorner y \urcorner a \urcorner a} \mid \ulcorner a \mid \underline{x \ulcorner b \ulcorner z \ulcorner a \smile} \mid b \urcorner \mid z \urcorner \mid \underline{\smile b^{\dagger} y \smile} \mid \ulcorner x \mid \ulcorner y \\
&\leftrightarrow \underline{\ulcorner x \smile y \urcorner a \urcorner a} \mid \ulcorner a \mid \underline{x \ulcorner b \ulcorner z \ulcorner a \smile} \mid b \urcorner \mid z \urcorner \mid \underline{\smile b^{\dagger} y \smile} \mid \ulcorner y \mid x \urcorner \\
&\leftrightarrow \underline{\ulcorner x \ulcorner y \smile a \urcorner a} \mid \ulcorner a \mid \underline{x \ulcorner b \ulcorner z \ulcorner a \smile} \mid b \urcorner \mid z \urcorner \mid \underline{\smile b^{\dagger} y \smile} \mid x \urcorner \mid y \urcorner \\
&\leftrightarrow \underline{\ulcorner x \ulcorner y \ulcorner a \smile a} \mid \underline{x \ulcorner b \ulcorner z \ulcorner a \smile} \mid b \urcorner \mid z \urcorner \mid \underline{\smile b^{\dagger} y \smile} \mid x \urcorner \mid y \urcorner \mid a \urcorner \\
&\leftrightarrow \underline{\ulcorner x \ulcorner y \ulcorner a \smile a} \mid \underline{x \ulcorner b \ulcorner z \smile a \urcorner} \mid b \urcorner \mid z \urcorner \mid \underline{\smile b^{\dagger} y \smile} \mid x \urcorner \mid y \urcorner \mid \ulcorner a \\
&\rightarrow \underline{\ulcorner x \ulcorner y \ulcorner a \ulcorner a} \mid \underline{x \ulcorner b \ulcorner z \smile a \urcorner} \mid b \urcorner \mid z \urcorner \mid \underline{\smile b^{\dagger} y \smile} \mid x \urcorner \mid y \urcorner \\
&\rightarrow \underline{x \ulcorner b \ulcorner z \smile a \urcorner} \mid b \urcorner \mid z \urcorner \mid \underline{\smile b^{\dagger} y \smile} \mid x \urcorner \mid y \urcorner \\
&\leftrightarrow \underline{x \ulcorner b \smile z \urcorner a \urcorner} \mid b \urcorner \mid \underline{\smile b^{\dagger} y \smile} \mid x \urcorner \mid y \urcorner \mid \ulcorner z \\
&\leftrightarrow \underline{x \smile b \urcorner z \urcorner a \urcorner} \mid \underline{\smile b^{\dagger} y \smile} \mid x \urcorner \mid y \urcorner \mid \ulcorner z \mid \ulcorner b \\
&\rightarrow \underline{x \urcorner b \urcorner z \urcorner a \urcorner} \mid \underline{\smile b^{\dagger} y \smile} \mid y \urcorner \mid \ulcorner z \mid \ulcorner b \\
&\rightarrow \underline{\smile b^{\dagger} y \smile} \mid y \urcorner \mid \ulcorner z \mid \ulcorner b \\
&\rightarrow \ulcorner z.
\end{aligned}$$

Hence

$$(J_{xyaz} \mid \ulcorner x \mid \ulcorner y)^n \rightarrow^* \ulcorner z^n$$

by induction,

$$(J_{xyaz})^n \mid \ulcorner x^n \mid \ulcorner y^n \rightarrow^* \ulcorner z^n$$

by associativity,

$$(\nu a)((J_{xyaz})^n \mid \ulcorner x^n \mid \ulcorner y^n) \rightarrow^* (\nu a) \ulcorner z^n$$

by isolation, and

$$J^n_{xyz} \mid \ulcorner x^n \mid \ulcorner y^n \rightarrow^* \ulcorner z^n$$

by ν-equivalence and the definition of J^n_{xyz}. □

4.2. *Will-correctness for populations*

We will now consider the difficulties involved in proving more interesting properties. We would like a transducer, for example, to work correctly in 'all possible contexts' so that after proving its correcness we could use it freely in any larger circuit without worrying about unwanted interferences. Unfortunately, this is just not true, because some context could absorb the $y\urcorner$ strand, which is public, and thus interfere with the transducer. We would have to consider instead 'all possible contexts that do not interfere with $y\urcorner$'. This is a rather awkward notion since for compositionality, for each component, we would need to keep track of all the elements in the context that the component might be interfering with. Moreover, the transducer interferes with $y\urcorner$, so it also interferes with (another copy or another population of) itself.

Let us consider a simpler 'progress' property: that the transducer does not deadlock with itself. This can be expressed as a will-correctness property: that for any intermediate state U, if $T^n_{xy} \mid tx^n \rightarrow^* U$, then $U \rightarrow^* ty^n$. This appears to require an induction on

https://doi.org/10.1017/S0960129512000102 Published online by Cambridge University Press

all possible intermediate configurations U for any n. Even for a fixed small n, the state space U can grow very large, which requires the use of automated state exploration tools. Note also that an induction on the length of $\rightarrow^*$ is problematic because of the reversible exchange rule: infinite sequences of reductions exist in almost all systems. In a stochastic interpretation of reduction, actual convergence can often be achieved (with measure 1), and this is another challenging property to prove. Here we simply illustrate how to check a will-correctness property, for a single copy of a transducer:

Proposition 4.5 (T^1_{xy} will-correctness). We have

$$T^1_{xy} \mid \ulcorner x \rightarrow^{\forall} \ulcorner y.$$

Moreover, $\ulcorner y$ is the only reachable terminal state.

Proof. We show that if $T^1_{xy} \mid \ulcorner x \rightarrow^* U$, then $U \rightarrow^* \ulcorner y$. We enumerate all distinct states U, up to algebraic equality, arising from $T^1_{xy} \mid tx$ by all possible traces, and then check that each state can lead to $\ulcorner y$. We assume $x \neq y$; indentation in the following represents a branch in the derivation, with one continuation to the right and one continuation down at the same level of indentation.

$$
\begin{array}{lll}
01. & (\nu a)\ \underline{\smile x \urcorner a \urcorner a} \mid \ulcorner a \mid \underline{x \ulcorner y \ulcorner a \smile} \mid y\urcorner \mid \ulcorner x & \\
02. & \leftrightarrow (\nu a)\ \underline{\ulcorner x \smile a \urcorner a} \mid \ulcorner a \mid \underline{x \ulcorner y \ulcorner a \smile} \mid y\urcorner \mid x\urcorner & \\
03. & \leftrightarrow (\nu a)\ \underline{\ulcorner x \ulcorner a \smile a} \mid \underline{x \ulcorner y \ulcorner a \smile} \mid y\urcorner \mid x\urcorner \mid a\urcorner & \\
04. & \leftrightarrow (\nu a)\ \underline{\ulcorner x \ulcorner a \smile a} \mid \underline{x \ulcorner y \smile a \urcorner} \mid y\urcorner \mid x\urcorner \mid \ulcorner a & \\
05. & \quad \leftrightarrow (\nu a)\ \underline{\ulcorner x \ulcorner a \ulcorner a} \mid \underline{x \ulcorner y \smile a \urcorner} \mid y\urcorner \mid x\urcorner & \\
06. & \quad\quad \rightarrow (\nu a)\ \underline{x \ulcorner y \smile a \urcorner} \mid y\urcorner \mid x\urcorner & \\
07. & \quad\quad \leftrightarrow (\nu a)\ \underline{x \smile y \urcorner a \urcorner} \mid x\urcorner \mid \ulcorner y & \\
08. & \quad\quad \leftrightarrow (\nu a)\ \underline{x\urcorner y \urcorner a \urcorner} \mid \ulcorner y & \\
09. & \quad\quad \rightarrow \ulcorner y & \\
10. & \quad \leftrightarrow (\nu a)\ \underline{\ulcorner x \ulcorner a \ulcorner a} \mid \underline{x \smile y \urcorner a \urcorner} \mid x\urcorner \mid \ulcorner y & \rightarrow 07 \\
11. & \quad \leftrightarrow (\nu a)\ \underline{\ulcorner x \ulcorner a \ulcorner a} \mid \underline{x\urcorner y \urcorner a \urcorner} \mid \ulcorner y & \rightarrow 08 \\
12. & \quad \leftrightarrow (\nu a)\ \underline{\ulcorner x \ulcorner a \ulcorner a} \mid \ulcorner y & \rightarrow 09 \\
13. & \leftrightarrow (\nu a)\ \underline{\ulcorner x \ulcorner a \smile a} \mid \underline{x \smile y \urcorner a \urcorner} \mid x\urcorner \mid \ulcorner a \mid \ulcorner y & \leftrightarrow 10 \\
14. & \leftrightarrow (\nu a)\ \underline{\ulcorner x \ulcorner a \smile a} \mid \underline{x\urcorner y \urcorner a \urcorner} \mid \ulcorner a \mid \ulcorner y & \leftrightarrow 11 \\
15. & \leftrightarrow (\nu a)\ \underline{\ulcorner x \ulcorner a \smile a} \mid \ulcorner a \mid \ulcorner y & \leftrightarrow 12
\end{array}
$$

All other states (up to algebraic equality) can be reduced to these states by well-mixing. We can then check that all these states have a path to state 9. The case for $x = y$ is similar: the state graphs are the same because, as can be seen above, there is never both an x redex and a different y redex in the same state, and when two x signals or cosignals can be chosen, it does not matter which one is chosen, by well-mixing. □

4.3. *The wisdom of crowds*

When composing transducers, the may-correctness property

$$T^n_{xy} \mid T^n_{yz} \mid \ulcorner x^n \rightarrow^* \ulcorner z^n$$

https://doi.org/10.1017/S0960129512000102 Published online by Cambridge University Press

follows simply from Proposition 4.2, but even just the will-correctness property

$$T^1_{xy} \mid T^1_{yz} \mid \ulcorner x \rightarrow^{\forall}_{\Gamma} z$$

(including $x = z$ and $y = z$ and $x = y = z$) does not follow from Proposition 4.5, but requires the analysis of a product state space. For example, $T^1_{xy} \mid T^1_{yx}$ can absorb the inputs $\ulcorner x \mid \ulcorner y$ sequentially (converting $\ulcorner x$ to a second $\ulcorner y$ and then $\ulcorner y$ to $\ulcorner x$) or in parallel (each transducer starting to process an input before producing an output). In fact, consider the following transducer that uses a public 'a' domain instead of a private one, and therefore is prone to interference:

$$T_{xay} = \underline{\smile x \urcorner a \urcorner a} \mid \ulcorner a \mid \underline{x \ulcorner y \ulcorner a \smile} \mid y \urcorner.$$

T_{xay} by itself satisfies may and will-correctness as shown above for T^1_{xy}, and so does T_{yax}. But the two together do not satisfy the will-correctness property of just producing $\ulcorner x$ on input $\ulcorner x$ because the following 'crosstalk' derivation is possible, where in the third step $a\urcorner$ goes to the 'wrong' gate:

$$\begin{aligned}
T_{xay} \mid T_{yax} \mid \ulcorner x = {} & \underline{\smile x \urcorner a \urcorner a} \mid \ulcorner a \mid \underline{x \ulcorner y \ulcorner a \smile} \mid y \urcorner \mid \underline{\smile y \urcorner a \urcorner a} \mid \ulcorner a \mid \underline{y \ulcorner x \ulcorner a \smile} \mid x \urcorner \mid \ulcorner x \\
\leftrightarrow {} & \underline{\ulcorner x \smile a \urcorner a} \mid \ulcorner a \mid \underline{x \ulcorner y \ulcorner a \smile} \mid y \urcorner \mid \underline{\smile y \urcorner a \urcorner a} \mid \ulcorner a \mid \underline{y \ulcorner x \ulcorner a \smile} \mid x \urcorner \mid x \urcorner \\
\leftrightarrow {} & \underline{\ulcorner x \ulcorner a \smile a} \mid \underline{x \ulcorner y \ulcorner a \smile} \mid y \urcorner \mid \underline{\smile y \urcorner a \urcorner a} \mid \ulcorner a \mid \underline{y \ulcorner x \ulcorner a \smile} \mid x \urcorner \mid x \urcorner \mid a \urcorner \\
\leftrightarrow {} & \underline{\ulcorner x \ulcorner a \smile a} \mid \underline{x \ulcorner y \ulcorner a \smile} \mid y \urcorner \mid \underline{\smile y \urcorner a \urcorner a} \mid \ulcorner a \mid \underline{y \ulcorner x \smile a \urcorner} \mid x \urcorner \mid x \urcorner \mid \ulcorner a \\
\rightarrow {} & \underline{\ulcorner x \ulcorner a \ulcorner a} \mid \underline{x \ulcorner y \ulcorner a \smile} \mid y \urcorner \mid \underline{\smile y \urcorner a \urcorner a} \mid \ulcorner a \mid \underline{y \ulcorner x \smile a \urcorner} \mid x \urcorner \mid x \urcorner \\
\rightarrow {} & \underline{x \ulcorner y \ulcorner a \smile} \mid y \urcorner \mid \underline{\smile y \urcorner a \urcorner a} \mid \ulcorner a \mid \underline{y \ulcorner x \smile a \urcorner} \mid x \urcorner \mid x \urcorner \\
\leftrightarrow {} & \underline{x \ulcorner y \ulcorner a \smile} \mid y \urcorner \mid \underline{\smile y \urcorner a \urcorner a} \mid \ulcorner a \mid \underline{y \smile x \urcorner a \urcorner} \mid x \urcorner \mid \ulcorner x \\
\rightarrow {} & \underline{x \ulcorner y \ulcorner a \smile} \mid \underline{\smile y \urcorner a \urcorner a} \mid \ulcorner a \mid \underline{y \urcorner x \urcorner a \urcorner} \mid x \urcorner \mid \ulcorner x \\
\rightarrow {} & \underline{x \ulcorner y \ulcorner a \smile} \mid \underline{\smile y \urcorner a \urcorner a} \mid \ulcorner a \mid x \urcorner \mid \ulcorner x.
\end{aligned}$$

The last state is final (no further progress can be made), and is not just the expected $\ulcorner x$ (which can be obtained by a different derivation). Moreover, no $\ulcorner y$ is ever produced. The system is deadlocked in a state where the expected output $\ulcorner x$ has been produced, but many other active components have been left to interfere with future operations. However, that last state, if supplied with an additional $\ulcorner y$, then unblocks and reduces to just $\ulcorner y \mid \ulcorner x$. Hence, although

$$T_{xay} \mid T_{yax} \mid \ulcorner x \not\rightarrow^{\forall}_{\Gamma} x,$$

we have

$$T_{xay} \mid T_{yax} \mid \ulcorner x \mid \ulcorner y \rightarrow^{\forall}_{\Gamma} x \mid \ulcorner y$$

(a formal proof of this fact requires an analysis of 115 states and 410 transitions, but DSD (Phillips and Cardelli 2009) and PRISM (Hinton *et al.* 2006) can build the full state space of the system and show that it has a single terminal state corresponding to $\ulcorner x \mid \ulcorner y$).

Now consider a system with two pairs of transducers: $T^2_{xay} \mid T^2_{yax} \mid \ulcorner x^2$. One pair may deadlock as described above, but the other pair may proceed correctly, producing $\ulcorner y$ as an intermediate result. That $\ulcorner y$ can unblock the first pair, so that both pairs terminate correctly. This means that in practice a large population of such gates does not deadlock

https://doi.org/10.1017/S0960129512000102 Published online by Cambridge University Press

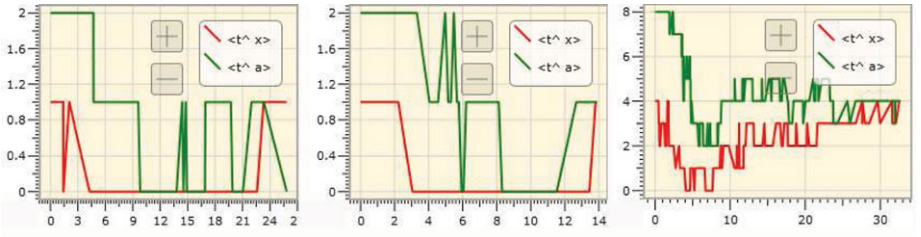


Fig. 17. (Colour online) Interfering transducers.

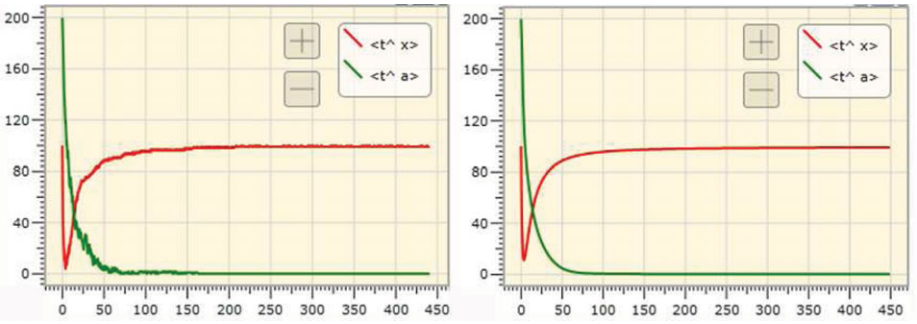


Fig. 18. (Colour online) The wisdom of crowds.

easily over an input population of $\ulcorner x$: each pair of stuck gates can be unblocked by another pair correctly producing a $\ulcorner y$, and it is very unlikely that a large fraction of gates ends up being blocked, as shown next.

The graphs in Figure 17 show individual runs of a stochastic simulation in DSD (Phillips and Cardelli 2009), with continuous time on the horizontal axis and the discrete number of copies on the vertical axis (all stochastic rates are set to 1.0). The left-hand graph is a simulation of one pair of transducers ($T_{xay} \mid T_{yax} \mid \ulcorner x$); this is a correct execution because when the simulation stops we are left with one $\ulcorner x$ (<t^ x> in the legend) and zero $\ulcorner a$ (<t^ a> in the legend). The middle graph shows an incorrect execution of the same system, where one $\ulcorner a$ (and other components not plotted) is left at the end. The right-hand graph shows the simulation of a system with four pairs of transducers ($T^4_{xay} \mid T^4_{yax} \mid \ulcorner x^4$) in the already *rare* case where all four pairs have deadlocked, leaving four copies of $\ulcorner a$. If we take a larger population of gates ($T^{100}_{xay} \mid T^{100}_{yax} \mid \ulcorner x^{100}$ in the left-hand graph of Figure 18), we get that virtually all runs execute correctly or nearly so. This can also be verified by numerical simulation of the corresponding ordinary differential equation system (the right-hand graph of Figure 18, which is obtained by selecting a different simulation option in DSD, where the vertical axis is now continuous and represents concentrations), showing that the concentration of deadlocking gates, as indicated by $\ulcorner a$, tends asymptotically to zero.

This 'wisdom of crowds' phenomenon, where wrong designs perform correctly in practice, is an interesting system property that is worth analysing and possibly exploiting. Gate designs that are 'logically wrong' due to interference may deadlock in small

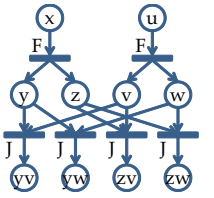
https://doi.org/10.1017/S0960129512000102 Published online by Cambridge University Press

populations, but large populations (of the kind usually found in chemical systems) may still converge to an almost-correct solution with high probability. Since the logically wrong designs are often more economical, there may be reasons for using them. For relatively small systems, the exact probability of correct termination (at a given time point) can be computed automatically by probabilistic modelchecking, using tools such as PRISM (Hinton *et al.* 2006). Otherwise, asymptotic behaviour can be investigated using stochastic or ODE simulation as shown above.

5. Testing

Gate and circuit designs have been tested with the DSD tool (Phillips and Cardelli 2009). We will give a simple example here by testing a combination of two fork and four join gates in the following configuration, where *yv*, *yw*, *zv*, *zw* are four output domains (that is, *yv* does not mean *y*.*v* in this section):

$F^n_{u,v,w}$ |
$J^n_{y,v,yv}$ |
$J^n_{y,w,yw}$ |
$J^n_{z,v,zv}$ |
$J^n_{z,w,zw}$ |
$F^n_{x,y,z}$ |
$\ulcorner x^m$ | $\ulcorner u^m$ (input, $m \leqslant n$)



The DSD script for this example is listed in Figure 19. Since fork and join gates accept inputs and produce outputs in a specific order, one should not expect identical rates of production of *yv*,*yw*,*zv*,*zw*. (If desired, we could mix populations of symmetric gates to achieve symmetric behaviour.) Figure 20 shows the results of an ordinary differential equations simulation with unit rates for toehold binding and unbinding, and with concentrations of 1.0 for the input signals and 10.0 for the gates; hence 10% of each gate is consumed during the computation. The system has a total of 54 single strand species, 108 double strand species and 172 reactions, and therefore 162 ODEs.

At time 3 (left-hand graph), *yv* is ahead of the other gates, with *zw* trailing last. At time 300 (right-hand graph) the computation has reached 90% completion with similar output quantities approaching the expected 0.5 concentration. One can use the tool further to examine the trajectories of all the species in the system to check that no deadlock occurs, and that all the structures are turned to output or waste.

6. Conclusions

We have shown how to implement fork and join gates using simple two-domain structures, and how to implement them in a 'clean' way that automatically removes all active garbage. We have provided a formal framework where we can perform calculations and study such questions, and we have discussed some simple correctness definitions and some complex behavioural properties. A formal proof of absence of gate interference under all possible combinations and numbers of gates and inputs will require an extensive amount of case analysis, which will probably require automatation, as well as the identification of

https://doi.org/10.1017/S0960129512000102 Published online by Cambridge University Press

```
directive sample 300.0 1000
directive plot <t^ yv>; <t^ yw>; <t^ zv>; <t^ zw>
new t@1.0,1.0

def T(N, x, y) = new a
( N * <t^ a> | N * <y t^>
| N * t^*:[x t^]:[a t^]:[a]
| N * [x]:[t^ y]:[t^ a]:t^* )

def C(N, x, y, z) = new a
( N * <t^ a> | N * <z t^>
| N * t^*:[x t^]:[y t^]:[a t^]:[a]
| N * [x]:[t^ z]:[t^ y]:[t^ a]:t^* )

def F(N, x, y, z) = new a
( N * <t^ a> | N * <y t^> | N * <z t^>
| N * t^*:[x t^]:[a t^]:[a]
| N * [x]:[t^ z]:[t^ y]:[t^ a]:t^* )

def J(N, x, y, z) = new a new b
( N * <t^ a> | N * <b t^> | N * <z t^>
| N * t^*:[x t^]:[y t^]:[a t^]:[a]
| N * [x]:[t^ b]:[t^ z]:[t^ a]:t^*
| N * t^*:[b y]:t^* )

( F(10, x, y, z) | F(10, u, v, w) | J(10, y, v, yv)
| J(10, y, w, yw) | J(10, z, v, zv) | J(10, z, w, zw)
| 1 * <t^ x> | 1 * <t^ u> )
```

Fig. 19. DSD script for fork/join circuit. This script can be run from a browser in DSD version 0.13 with 'deterministic' simulation (see Phillips and Cardelli (2009); see http://lepton.research.microsoft.com/webdna for the latest DSD). For reference, it contains the parametric definitions of transducer gates (T, Figure 3), catalytic gates (C, Figure 7), fork gates (F, Figure 6) and join gates (J, Figure 8), although it then uses only fork and join gates.

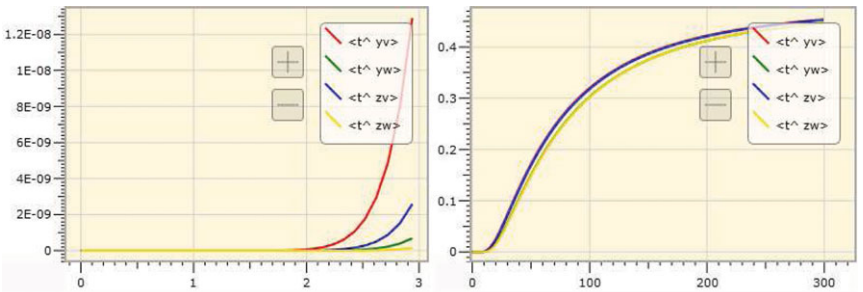


Fig. 20. (Colour online) Testing a fork/join circuit.

appropriate invariants. Alternatively, one may gain confidence in the designs by simulation testing.

Acknowledgments

Figures were prepared with the DSD tool (Phillips and Cardelli 2009). I would like to thank the members of the Molecular Programming Project at Caltech and the University

https://doi.org/10.1017/S0960129512000102 Published online by Cambridge University Press

of Washington for many tutorials and discussions, and an anonymous referee for raising the issues of reversible output release and related examples.

References

Benenson, Y., Paz-Elizur, T., Adar, R., Keinan, E., Livneh, Z. and Shapiro, E. (2001) Programmable and Autonomous Computing Machine made of Biomolecules. *Nature* **414** (22).

Berry, G. and Boudol, G. (1989) The Chemical Abstract Machine. *Proceedings 17th POPL*, ACM 81–94.

Cardelli, L. (2011) Strand Algebras for DNA Computing. *Natural Computing* **10** (1) 407–428.

Fontana, W. (2006) *Science* **314** (8).

Green, S. J., Lubrich, D. and Turberfield, A. J. (2006) DNA Hairpins: Fuel for Autonomous DNA Devices. *Biophysical Journal* **91** 2966–2975.

Hagiya, M. (2004) Towards Molecular Programming. In: Ciobanu, G. and Rozenberg, G. (eds.) *Modelling in Molecular Biology*, Springer-Verlag.

Hinton, A., Kwiatkowska, M., Norman, G. and Parker, D. (2006) PRISM: A tool for automatic verification of probabilistic systems. In: Hermanns, H. and Palsberg, J. (eds.) Proceedings 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06). *Springer-Verlag Lecture Notes in Computer Science* **3920** 441–444.

Milner, R. (1999) *Communicating and Mobile Systems: The π-Calculus*, Cambridge University Press.

Phillips, A. and Cardelli, L. (2009) A Programming Language for Composable DNA Circuits. *Journal of the Royal Society Interface* **6** S419–S436.

Qian, L. and Winfree, E. (2011) A simple DNA gate motif for synthesizing large-scale circuits. *Journal of the Royal Society Interface* doi: 10.1098/rsif.2010.0729.

Seelig, G., Soloveichik, D., Zhang, D. Y. and Winfree, E. (2006) Enzyme-Free Nucleic Acid Logic Circuits. *Science* **314** (8).

Soloveichik, D., Seelig, G. and Winfree, E. (2010) DNA as a Universal Substrate for Chemical Kinetics. *PNAS* **107** (12) 5393–5398.

Yin, P., Choi, H. M. T., Calvert, C. R. and Pierce, N. A. (2008) Programming biomolecular self-assembly pathways. *Nature* **451** 318–322.

Yurke, B. and Mills Jr., A. P. (2003) Using DNA to Power Nanostructures. *Genetic Programming and Evolvable Machines Archive* **4** (2) 111–122.

Zhang, D. Y. (2010) *Dynamic DNA strand displacement circuits*, Ph.D. Dissertation, California Institute of Technology. (Available at http://resolver.caltech.edu/CaltechTHESIS:05262010-173410602.)

Zhang, D. Y. (2011) *Journal of the American Chemical Society* **133** (4) 1077–1086.

Zhang, D. Y., Turberfield, A. J., Yurke, B. and Winfree, E. (2007) Engineering Entropy-driven Reactions and Networks Catalyzed by DNA. *Science* **318** 1121–1125.

Zhang, D. Y. and Seelig, G. (2011) Dynamic DNA nanotechnology using strand-displacement reactions. *Nature Chemistry* **3** 103–113.

Zhang, D. Y. and Winfree, E. (2009) Control of DNA Strand Displacement Kinetics Using Toehold Exchange. *Journal of the American Chemical Society* **131** (47) 17303–17314.

https://doi.org/10.1017/S0960129512000102 Published online by Cambridge University Press